

The *Mercury* System: Embedding Computation into Disk Drives

Roger D. Chamberlain, Ron K. Cytron, Mark A. Franklin, and Ronald S. Indeck
Center for Security Technologies, Washington University, St. Louis, Missouri
roger@ccrc.wustl.edu, cytron@cse.wustl.edu, jbf@ccrc.wustl.edu, rsi@ee.wustl.edu

1. Introduction

Having inexpensive data storage has enabled the amassing of vast amounts of information. At present, these data sets far exceed the capacity of modern processors, so searching them has become a serious challenge. In a recent invited talk at the High Performance Embedded Computing Workshop, John Reynnders of Celera Genomics commented that, “The size of the databases we deal with is no longer measured in terabytes, but in exabytes.” [1]

The *Mercury* system is a prototype data search engine that can be embedded within the disk drive itself. We focus on the specific problems associated with searches of unstructured, unindexed data. Three specific applications include approximate matching of text (important for text searches of specific interest to homeland security where the original alphabet is different than the Latin alphabet and transliteration is involved), genomics and proteomics searches (important biological applications), and image searches (also of significant interest for homeland security).

Currently, data searching applications are implemented using traditional, off-the-shelf hardware platforms. Figure 1 illustrates the relevant features of virtually all of these systems. A disk (actually many disks) is attached via a controller to the I/O bus of a computer system. A path (labeled “bridge” in the figure) exists that enables data to flow from the I/O bus to the memory bus (and therefore to the system’s main memory). When an algorithm is executed on the processor, references to the main memory cause the data to be loaded into cache, at which point the processor can efficiently access the data.

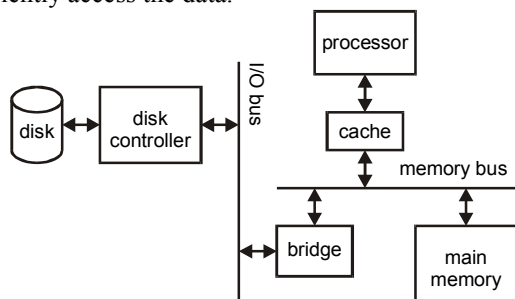


Figure 1. Typical hardware platform for data searching applications.

Add the operating system overhead to the above data movement requirements, and it is clear that there are significant data movement inefficiencies in current systems. Yet this is the system environment associated with development and deployment of virtually all of today’s data search applications. The result is that even though individual components in the system are quite fast (e.g., modern processors have clock speeds exceeding 2 GHz), the overall performance suffers because these fast components are used inefficiently.

Our system dramatically increases the speed with which large volumes of data can be searched by eliminating the above inefficiencies and searching data much closer to where it resides, on the disk, and by performing low-level search operations directly in reconfigurable hardware. In effect, the computation is being embedded into the drive.

2. Overall System Architecture

The *Mercury* system architecture is illustrated in Figure 2. Associated directly with a disk head is a Data Shift Register (DSR) that receives data streaming off the head at disk rotational speeds. The data in the DSR is made available (in parallel form) to reconfigurable logic that performs low-level searching operations on the data that has been retrieved off the disk. The specific function performed by the reconfigurable logic is tailored to the particular application of interest. Example functions for a set of applications are described below.

Also present in the system is a microprocessor that is used both for control duties (e.g., managing the function of the reconfigurable logic) and higher-level data searching operations. The remainder of the system reflects traditional designs, with an I/O bus, a bridge to the memory bus, and a classic memory hierarchy. The host processor is still responsible for managing the file system and maintaining the general functionality of the database, the new embedded hardware is used primarily for high-volume data searching operations.

In the description that follows, we assume (for discussion purposes) that the data is unstructured text and we are performing string matching queries. Realistic queries are generally compound in nature. In this case the search involves both matching the query strings to documents on the disk, and determining if the relationships exist when matches occur (or don’t occur).

Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE 20 AUG 2004		2. REPORT TYPE N/A		3. DATES COVERED -	
4. TITLE AND SUBTITLE The Mercury System: Embedding Computation into Disk Drives				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Center for Security Technologies, Washington University, St. Louis, Missouri				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release, distribution unlimited					
13. SUPPLEMENTARY NOTES See also ADM001694, HPEC-6-Vol 1 ESC-TR-2003-081; High Performance Embedded Computing (HPEC) Workshop(7th). , The original document contains color images.					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 37	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

The *Mercury* System: Embedding Computation into Disk Drives

Roger Chamberlain, Ron Cytron,
Mark Franklin, Ron Indeck

Center for Security Technologies
Washington University in St. Louis

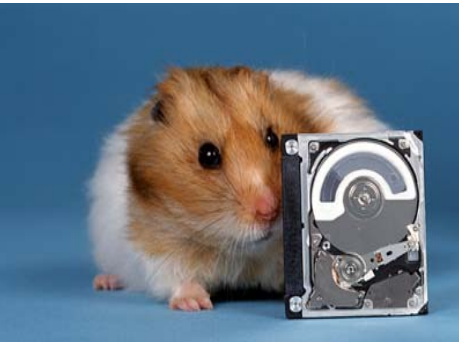
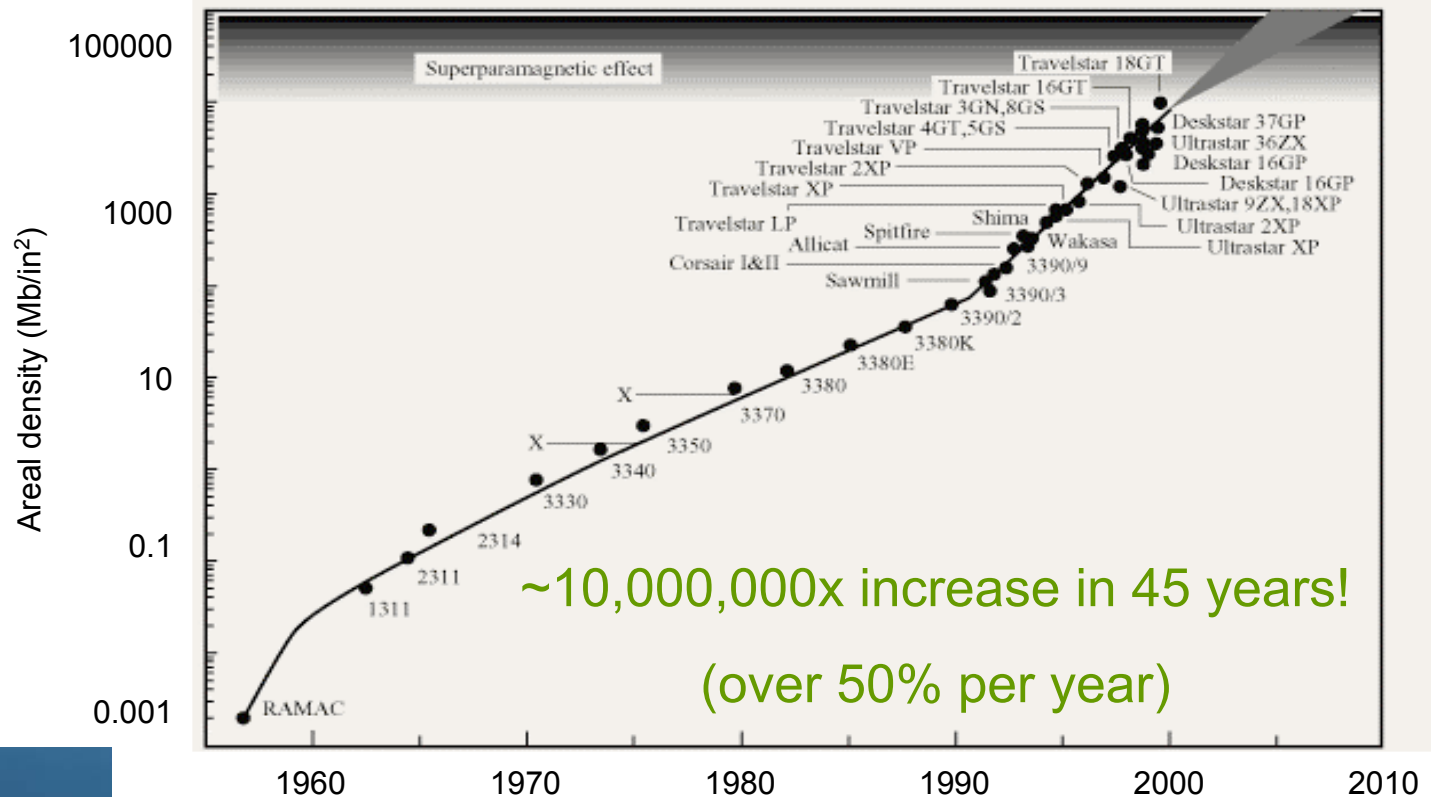


Center for Security
Technologies



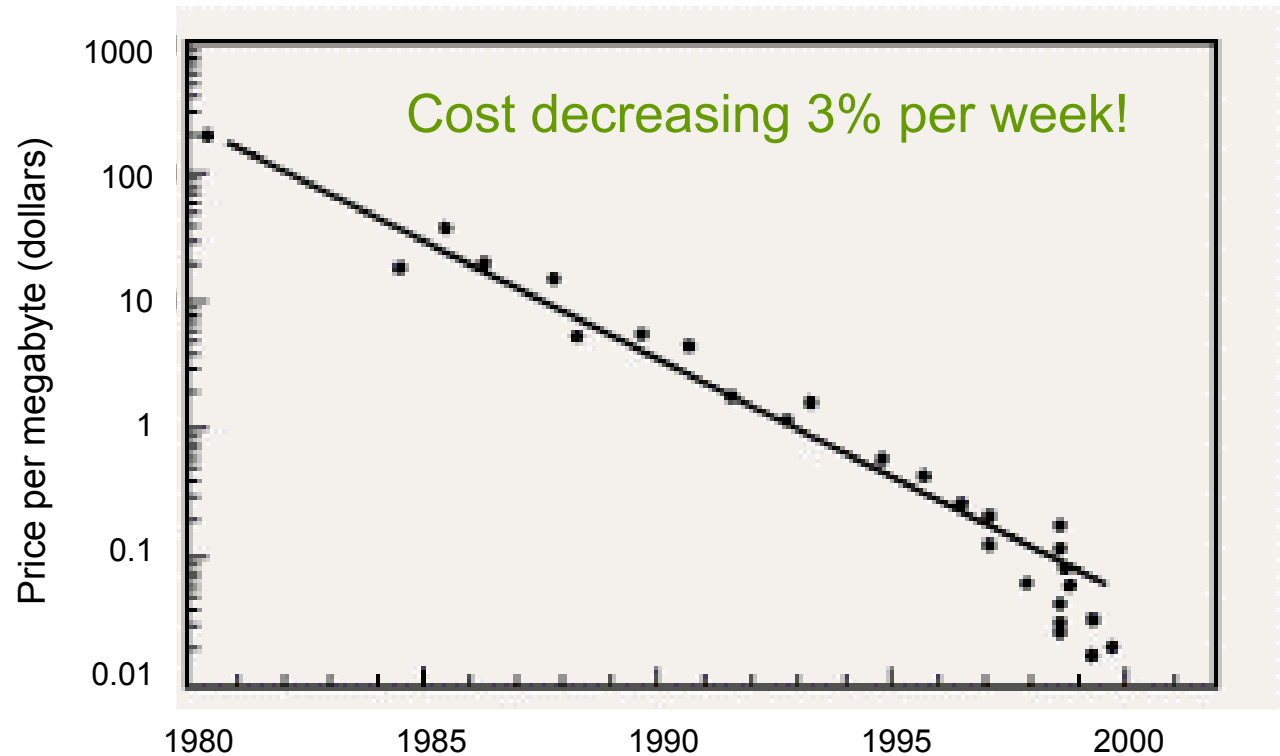
Washington University in St. Louis

Enabling Technology: Disk Drives



Magnetic disk storage areal density vs. year of IBM product introduction (From D. A. Thompson)

Cost per Megabyte



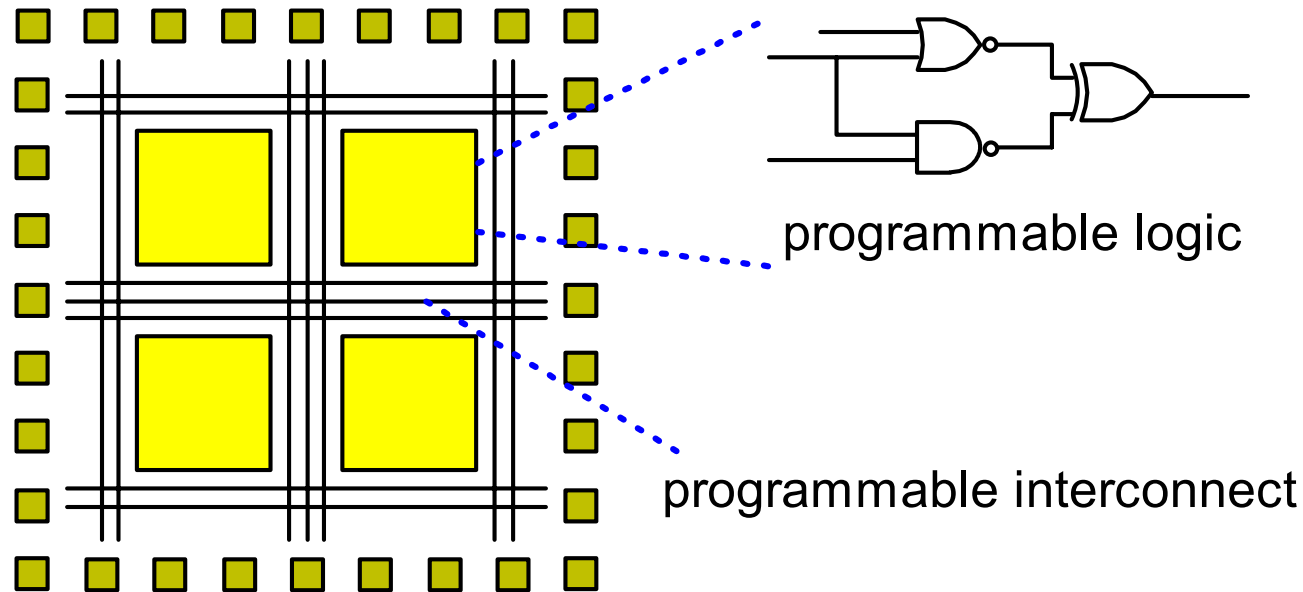
Price history of hard disk product vs. year of product introduction (From D. A. Thompson)

Massive Data

- Storage industry shipped 4,000,000,000,000,000,000 Bytes last year
- MasterCard recently installed a 200 TByte data warehouse in St. Louis
- US intelligence services collect data equaling the printed collection of the US Library of Congress every day!



Enabling Technology: Reconfigurable Hardware



- Field Programmable Gate Arrays (FPGAs) provide custom logic function capability
- Operate at hardware speeds
- Can be altered (reconfigured) in the field to meet specific application needs

What are we doing?

Within the Center, we are combining the capabilities of these two enabling technologies to build extremely fast data search engines.

We do this by moving the search closer to the data, and performing it in hardware rather than software.



Important Application: Intelligence Data



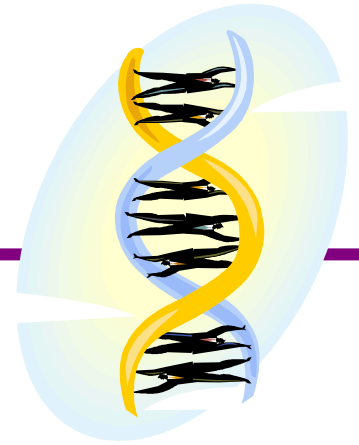
- Lots of data
 - Public (e.g., web pages)
 - Clandestine (e.g., via national technical means)
- Growing constantly
- Many perturbations of individual words
 - Tzar, Tsar, Czar, ...
- Query and field types aren't known *a priori*

Finding a needle in a haystack

- Text can contain errors
- Often seek an *approximate* match, e.g.
needle
- No match? Try 2-transpositions
enedle, needle, nedele, neelde, needel
- No match? Try 1-deletions
eedle, nedle, nedle, neele, neede, needl
- No match? Try insertions, larger edits, ...



Genome Application

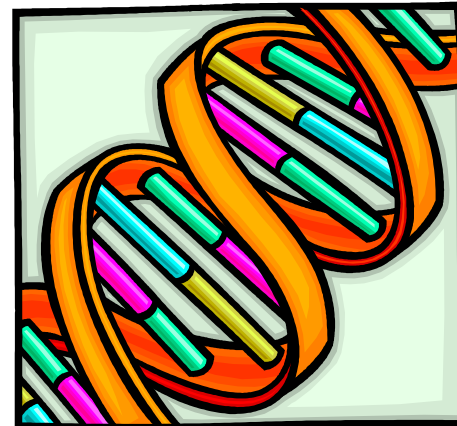


- Genome maps being expanded daily
 - 80,000 genes, 3 billion base pairs (A,C,G,T)
- Look for matches
 - Identify function
 - Disease: understand, diagnose, detect, therapy
 - Biofuels, warfare, toxic waste
 - Understand evolution
 - Forensics, organ donors, authentication
 - More effective crops, disease resistance



DNA String Matching

- Looking for *CACGTTAGT...TAGC*
- Interested in matches and near matches
- Search human genome, other gene oceans
 - Need to search entire data sets



Bio Computation Problem

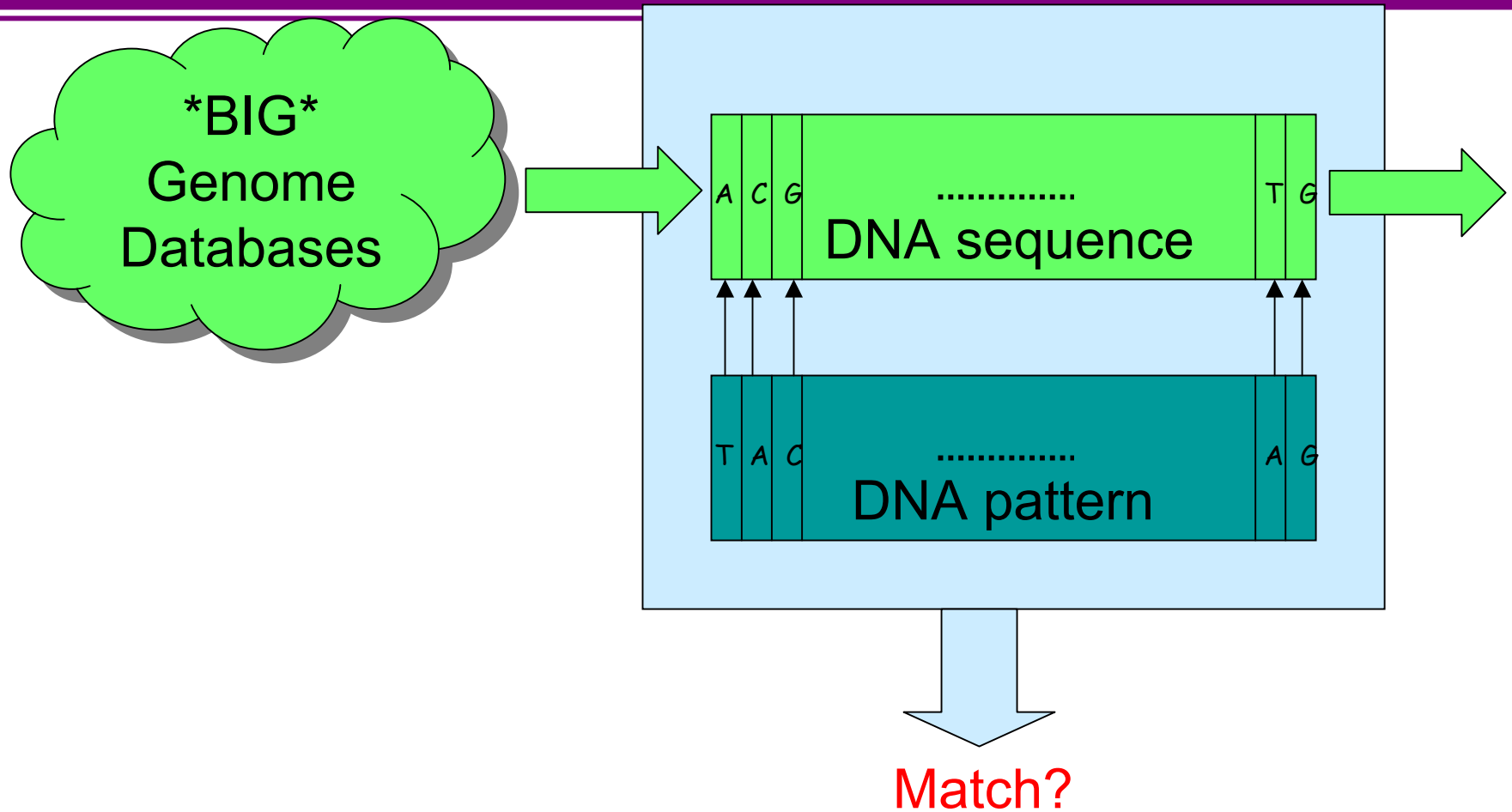


Image Database Applications

- Challenging database
- Unstructured
- Massive data sets
- Don't know what we need to look for in each picture



Object Recognition

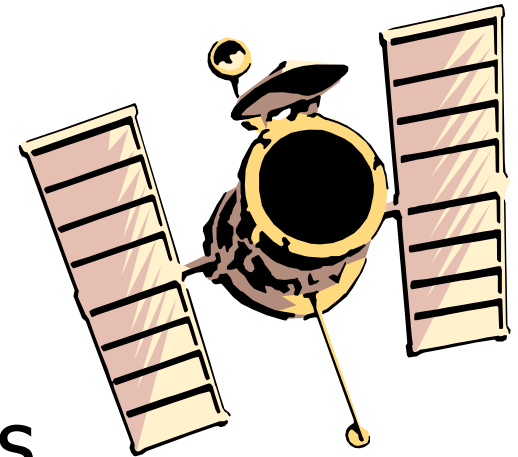
- Face recognition
- Match template with image
- Template database must be searched
- Strict time constraints for matching and overall search

Washington University Campus



Satellite Data

- Low orbit fly-over every 90 minutes
- Look for *differences* in images
 - Large objects
 - Troops
 - Changes to landscape
- Flag, transmit these differences immediately





How do we find what we're looking for most effectively?!



Conventional Structured Database

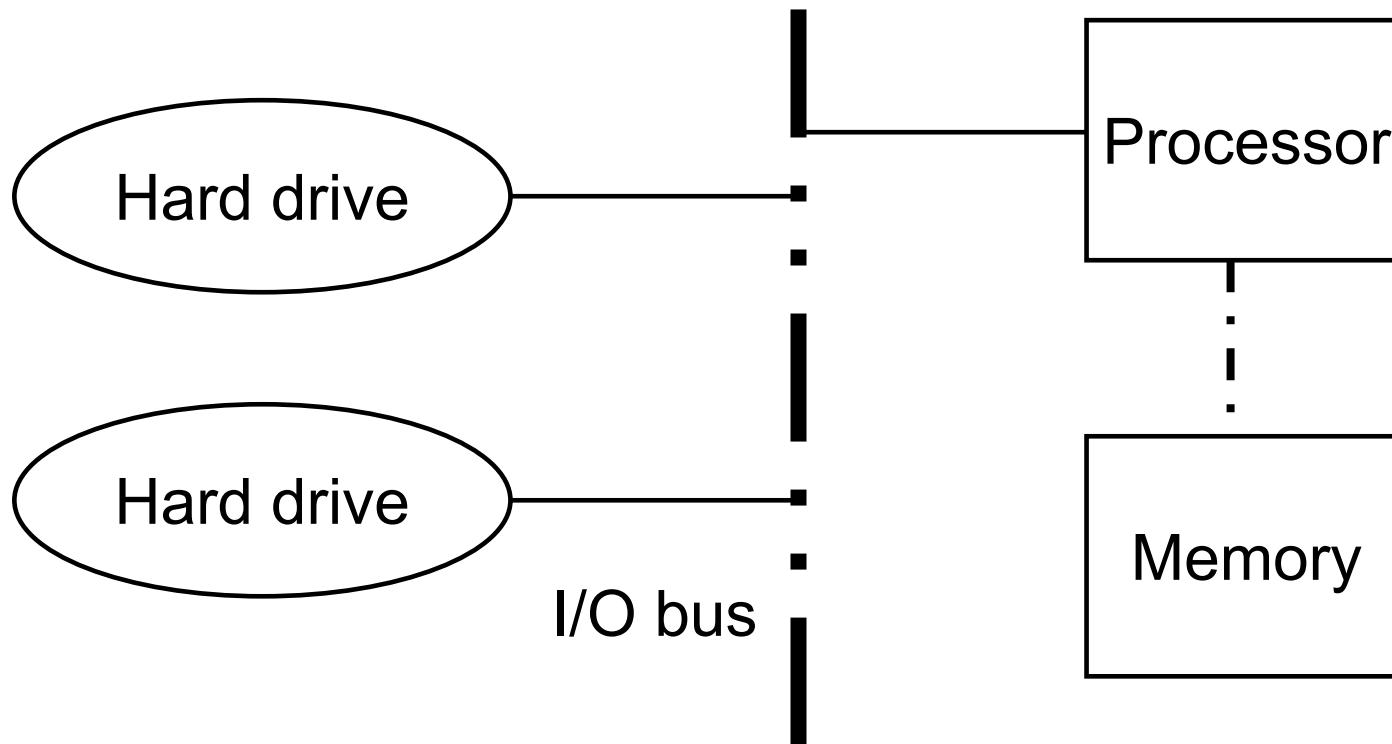
<i>D_{id}</i>	<i>Document</i>
1	Agent James Bond
2	Agent mobile computer
3	James Madison movie
4	James Bond movie

<i>Word</i>	<i>Inverted list - pointers</i>
agent	<1,2>
Bond	<1,4>
computer	<2>
James	<1,3,4>
Madison	<3>
mobile	<2>
movie	<3,4>

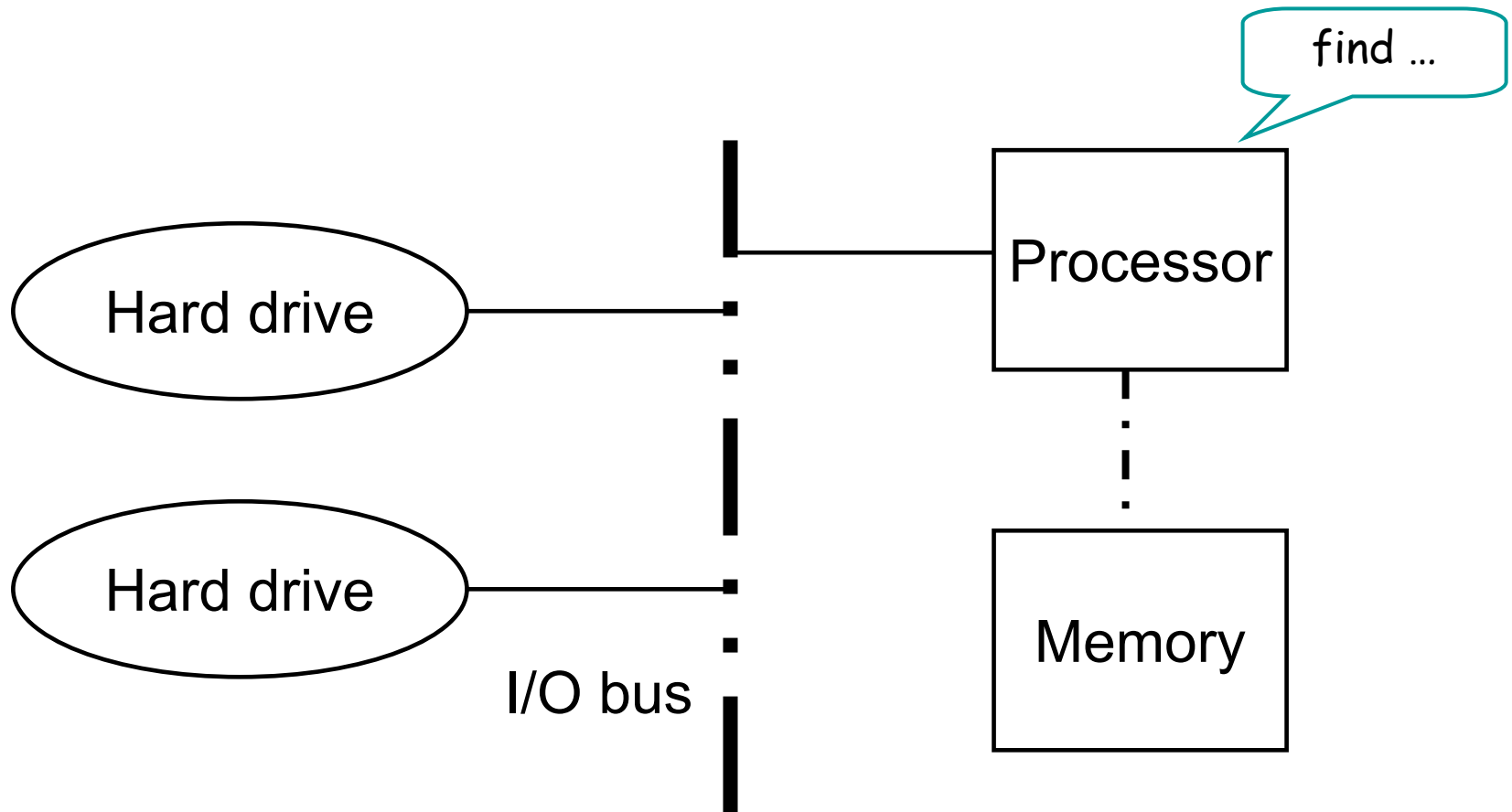
Challenges in Searching These Massive Databases

- If we know what we will be looking for
 - Need to build index beforehand
 - Maintain index as it changes
- If we ***don't*** know what we want *a priori*
 - Need to search the whole database!

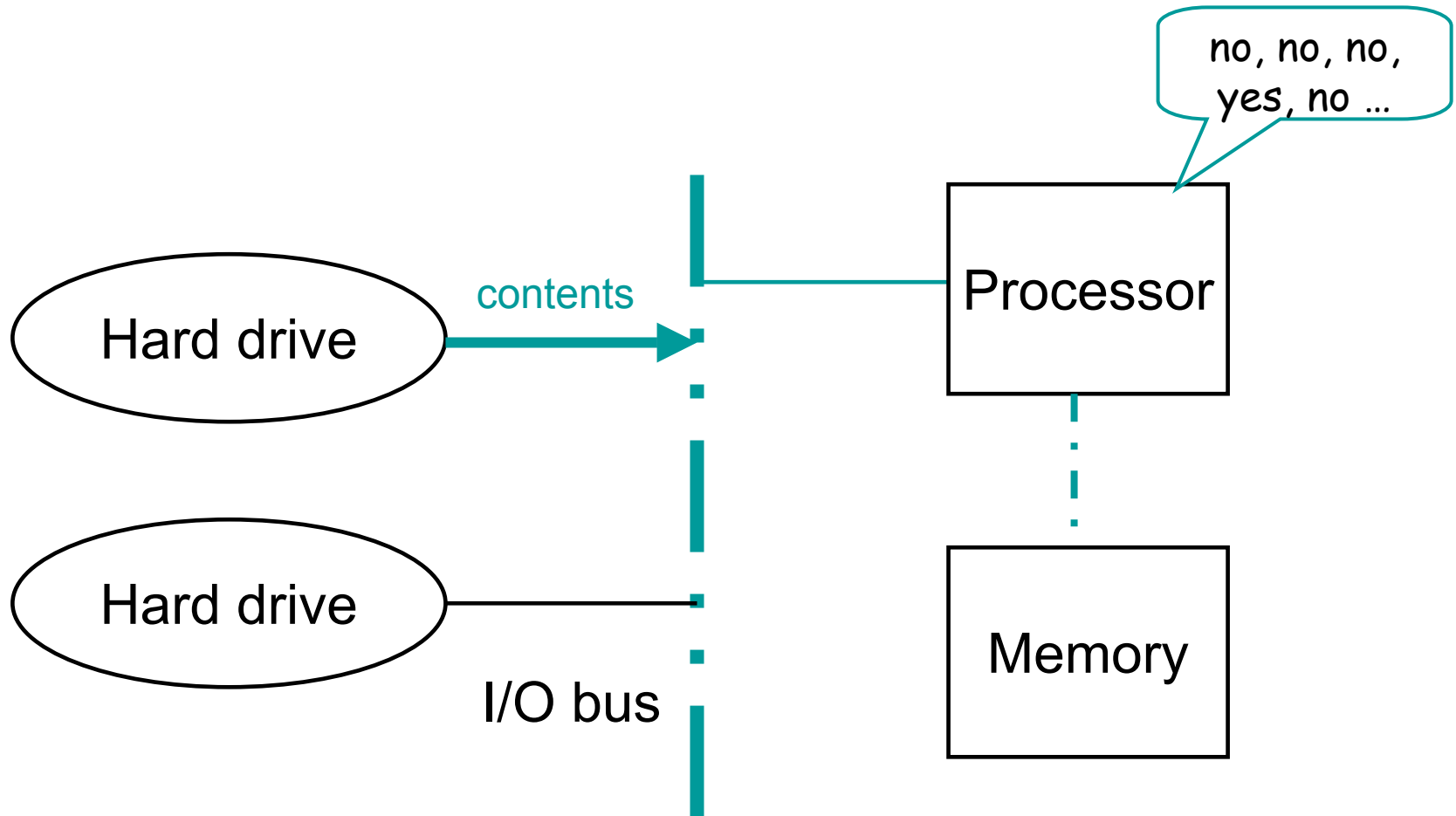
Conventional Search



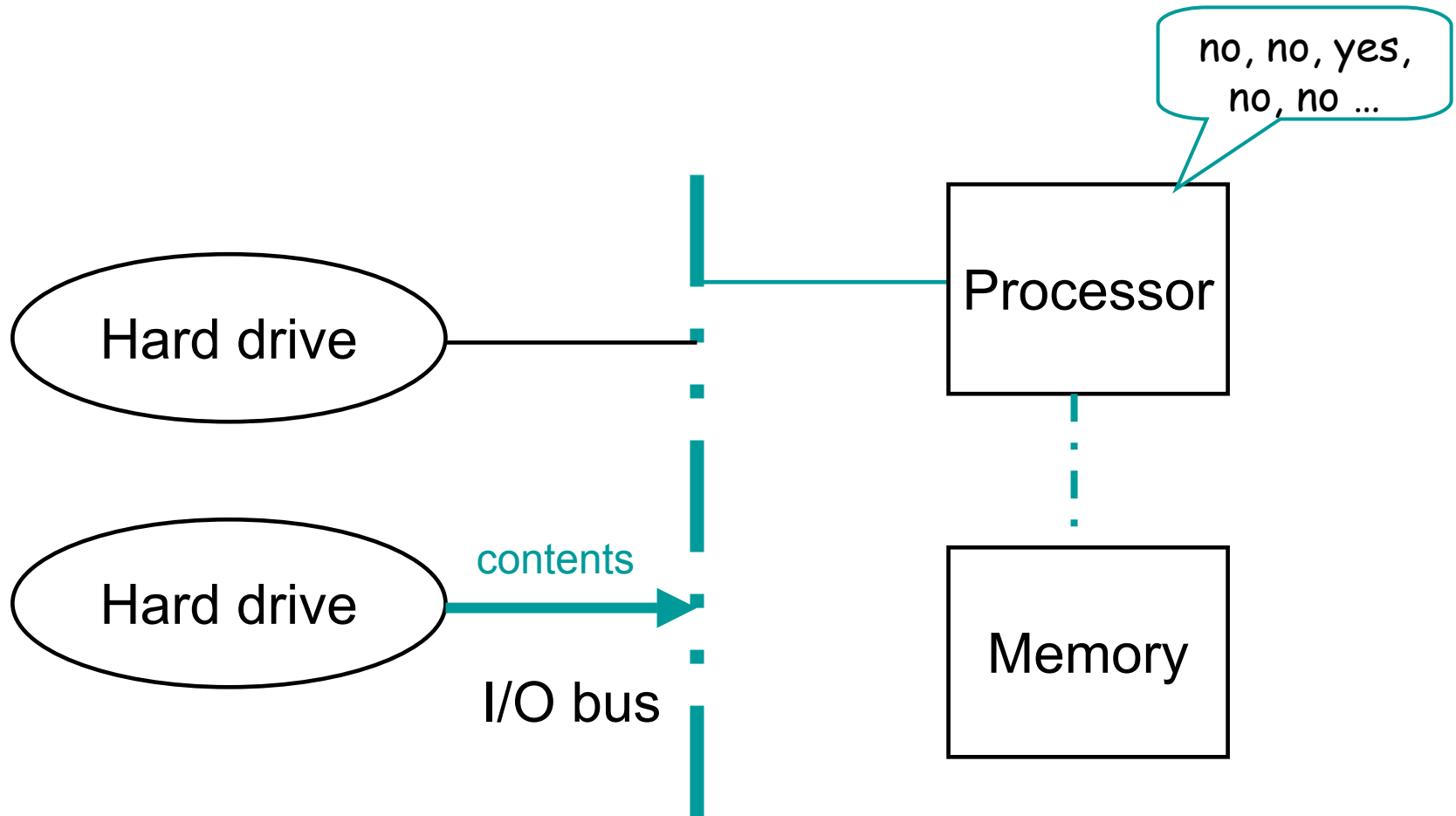
Conventional Search



Conventional Search



Conventional Search



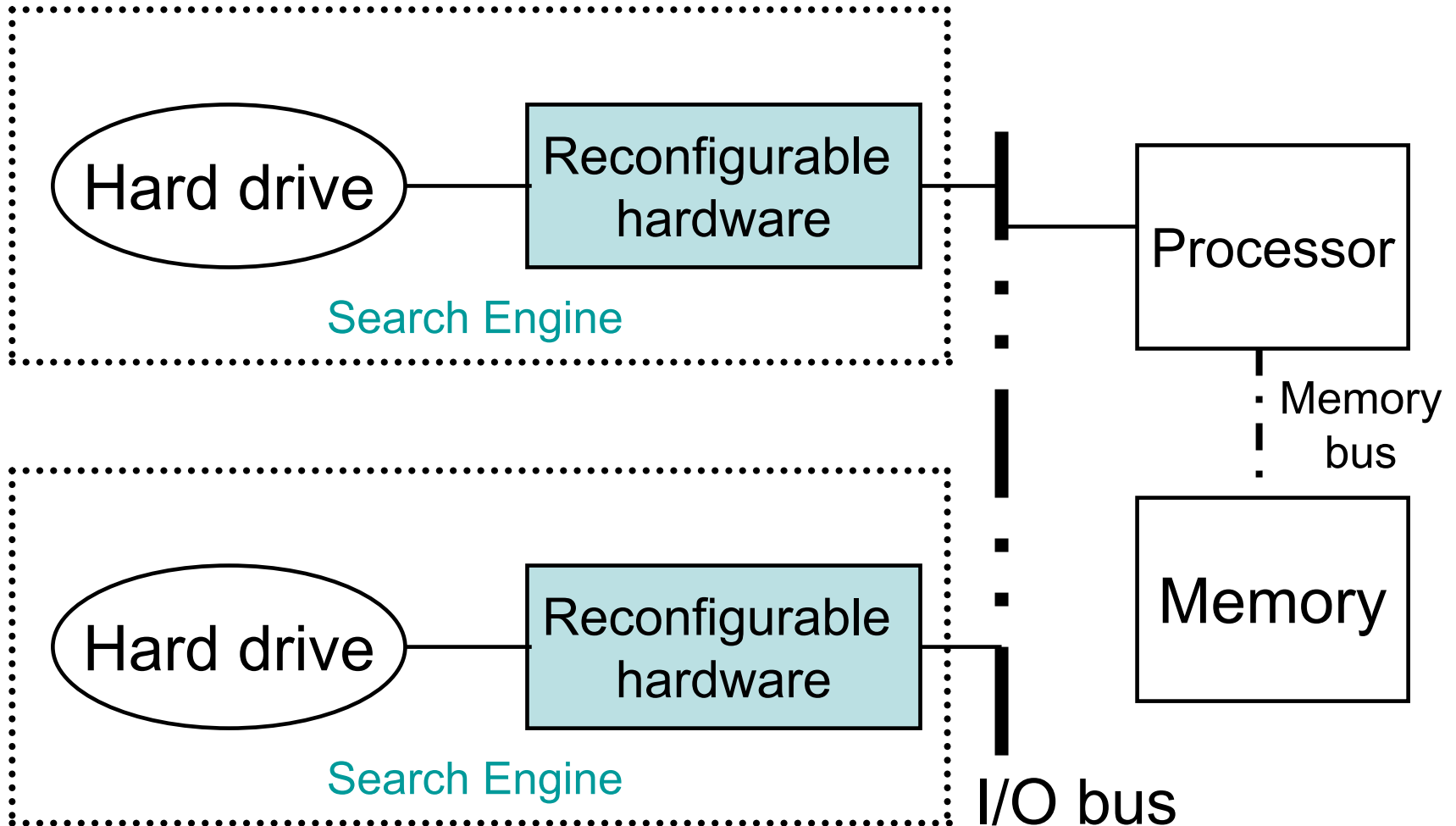
Conventional Approach



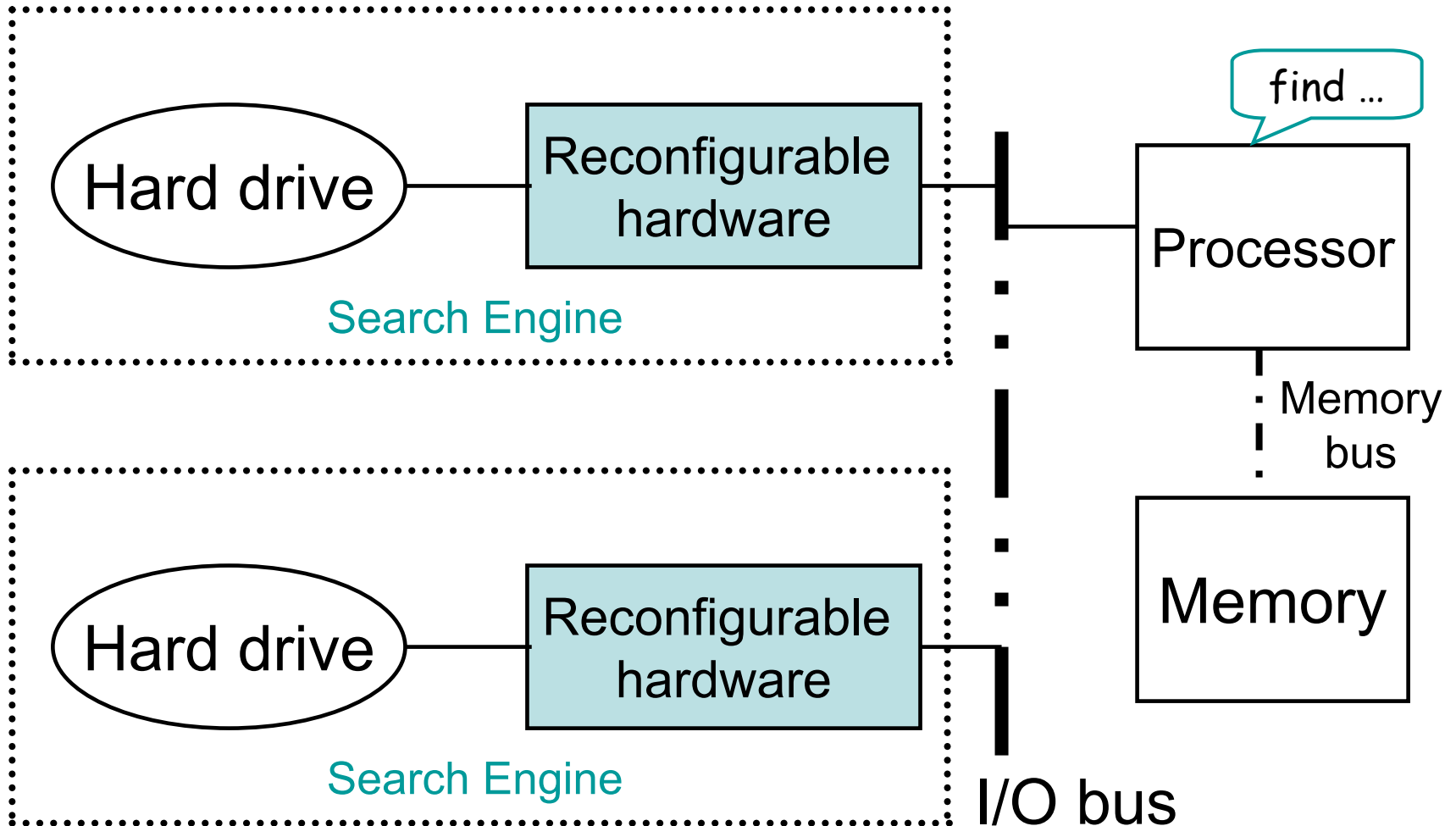
WUSTL's Approach



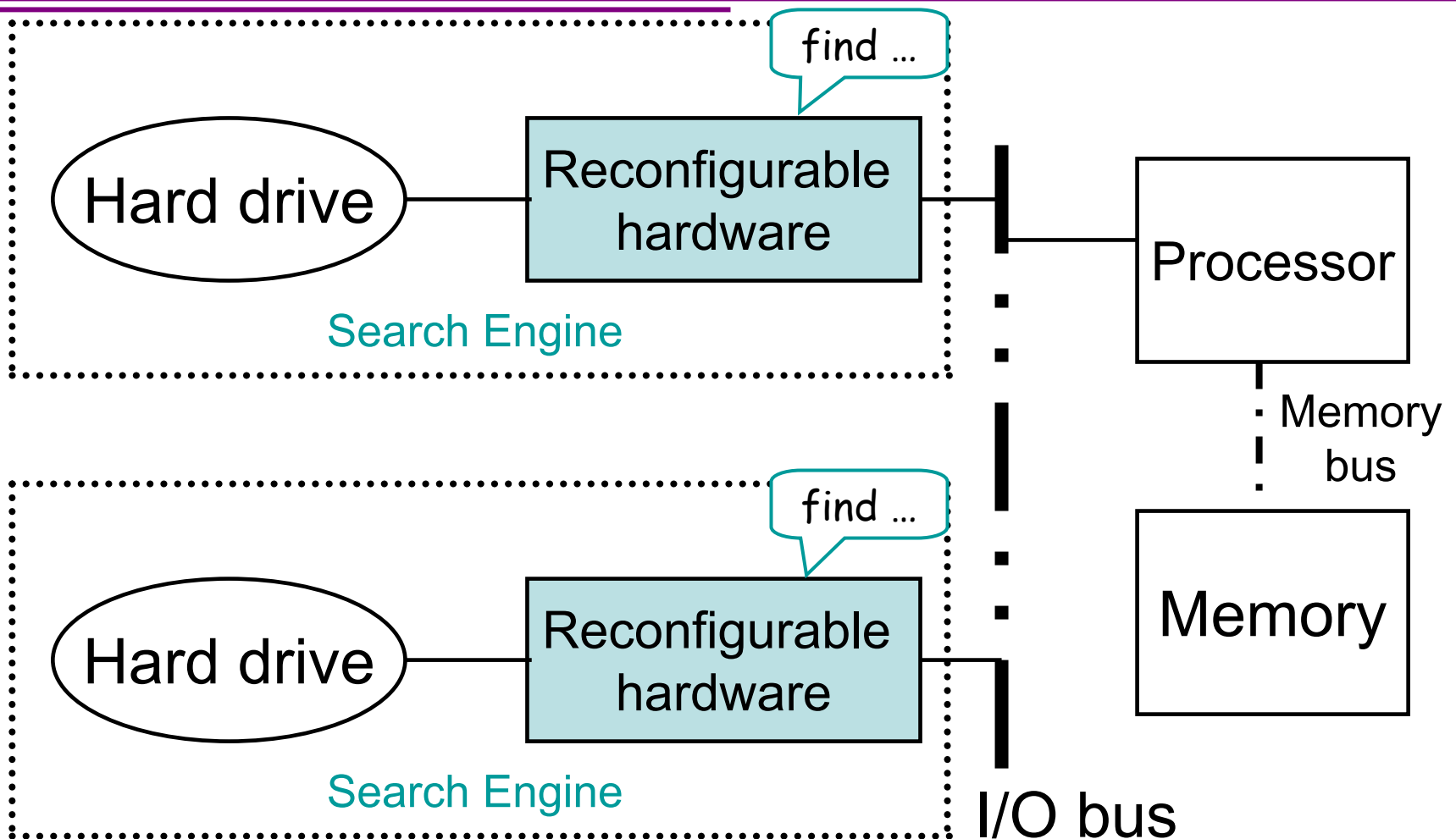
Streaming Approach



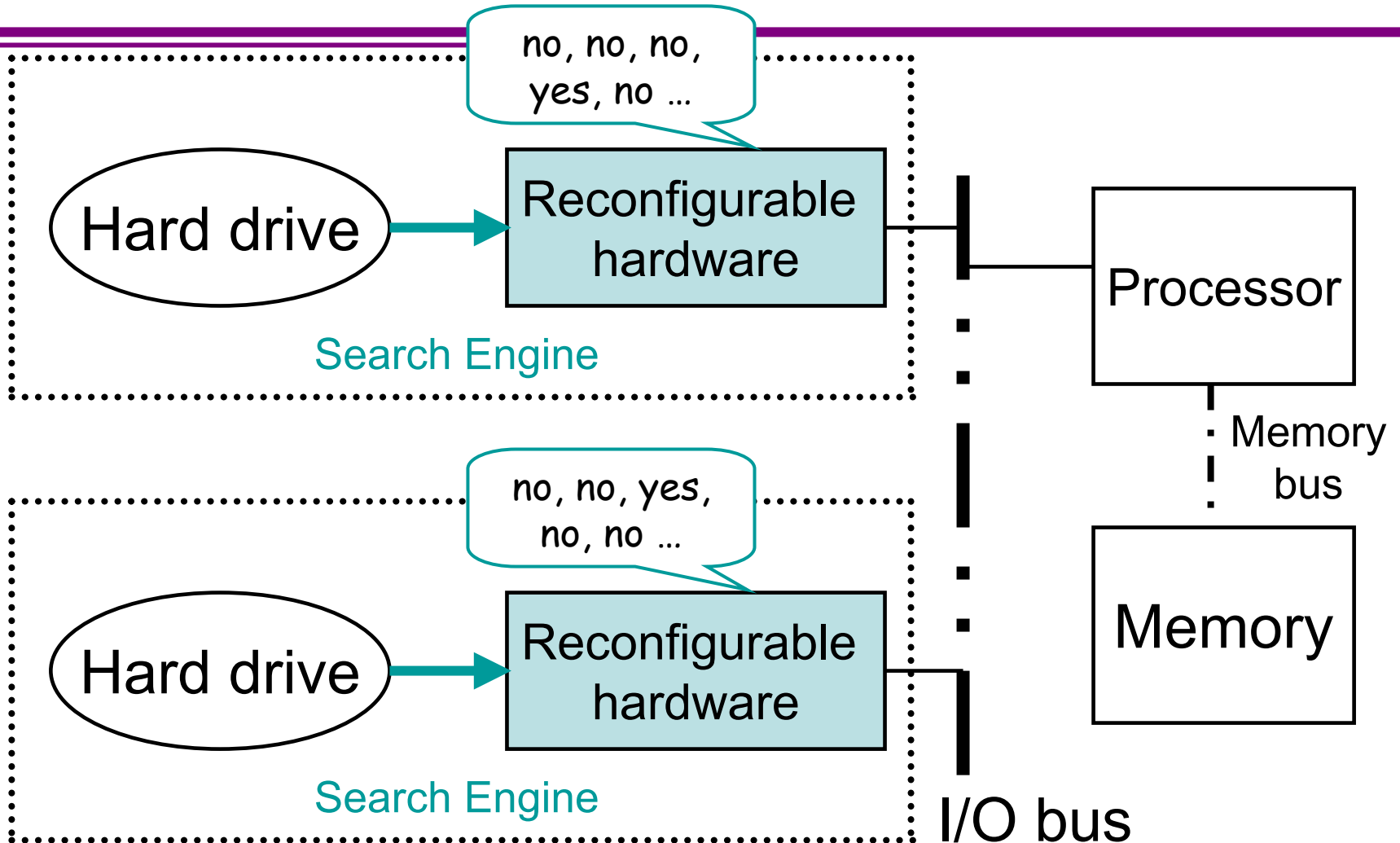
Streaming Approach



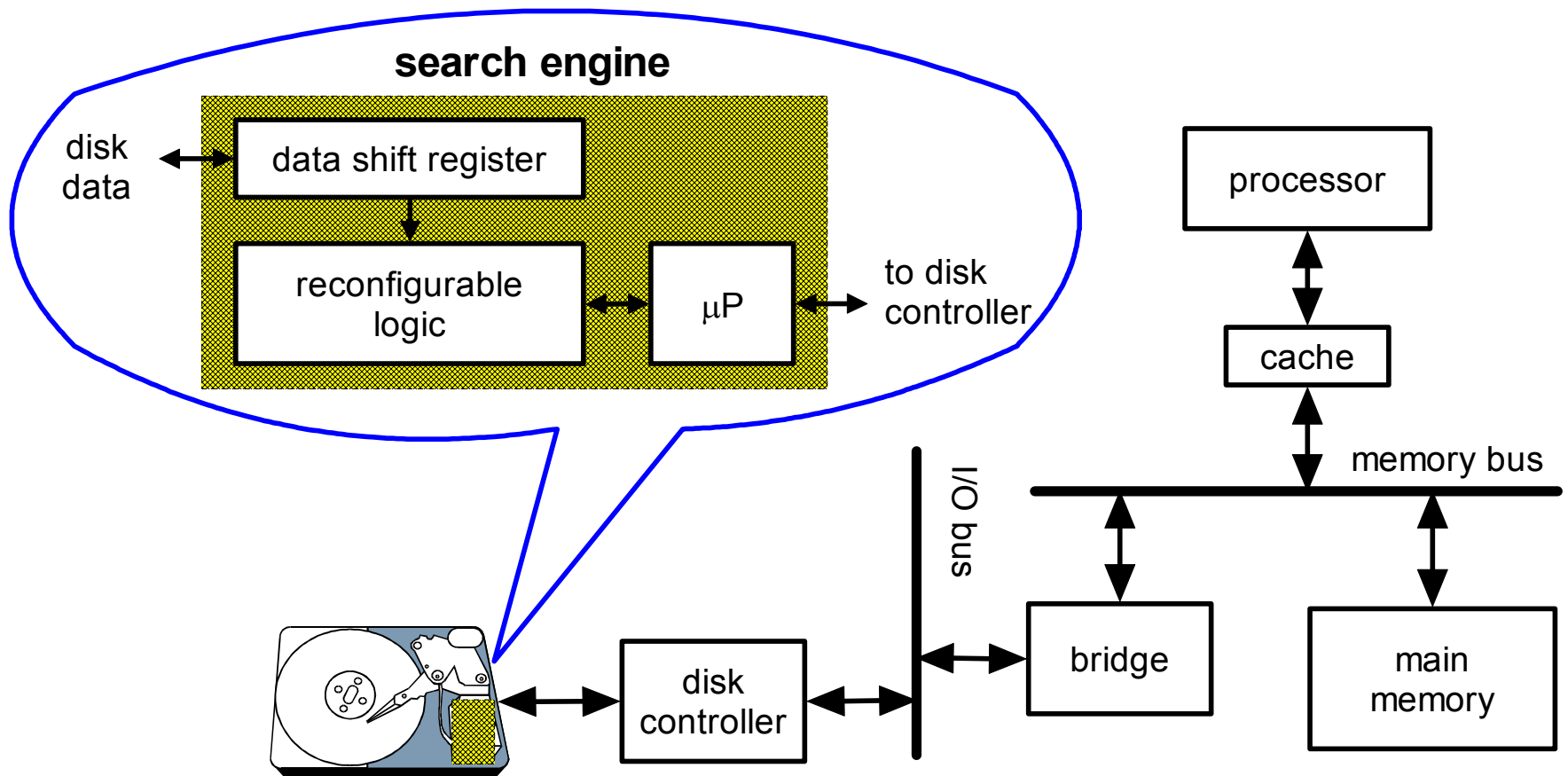
Streaming Approach



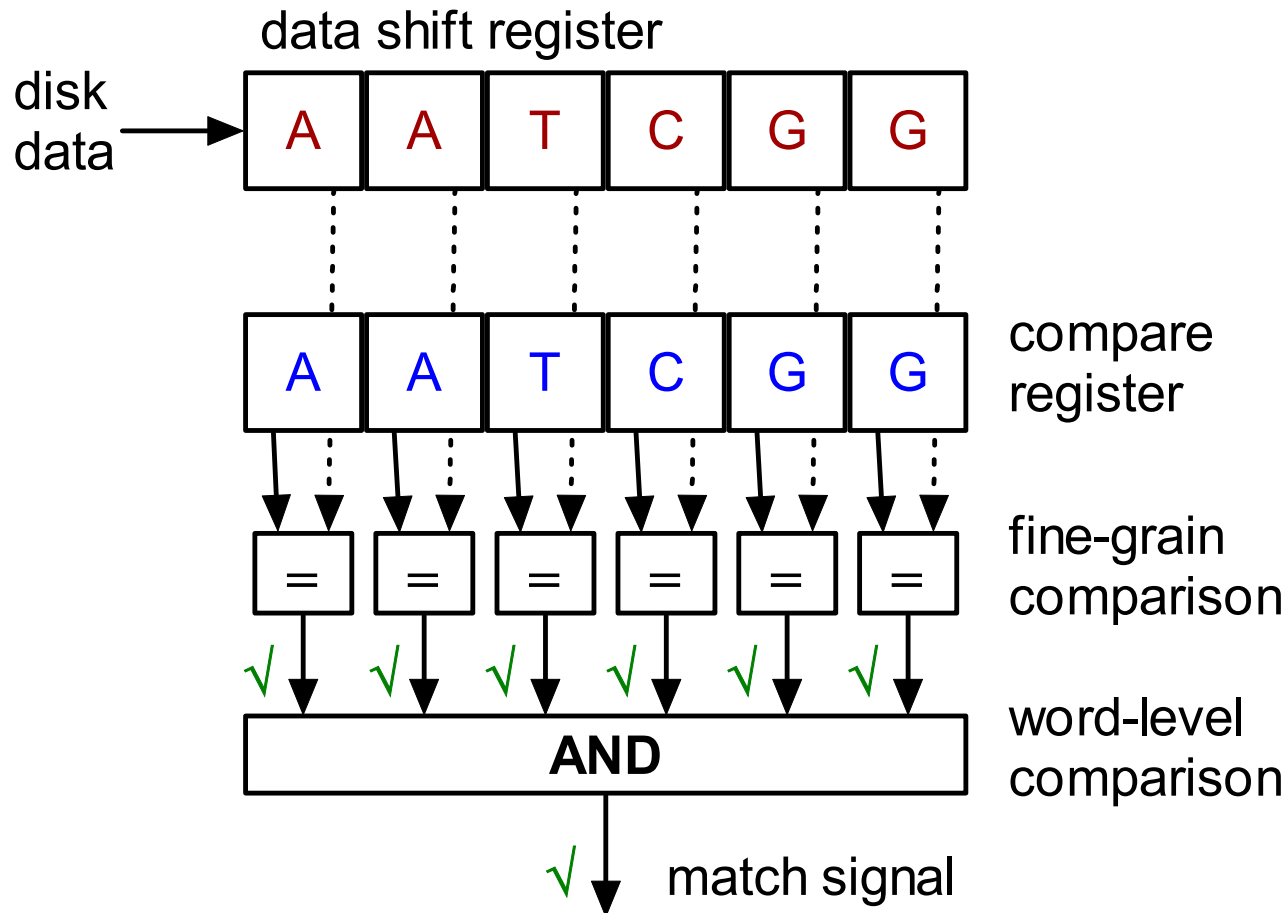
Streaming Approach



Search Engine in Context



Reconfigurable Hardware for Text Searches



Sources of Performance Gains

1. Disk Search Parallelism: Each engine searches in parallel across a disk or disk surface
2. System Parallelism: Searching is off-loaded to search engines and main processor can perform other tasks
3. Reduced data movement overhead: Disk data moves principally to search engine, not successively over system bus, memory bus, to cache, etc.
4. Hardware logic for searching: Searching, matching, and query operations are performed on streaming data in hardware rather than in software
5. Specialized hardware logic tailored to queries: Reconfigurable hardware permits matching the query logic to the search engine logic and preserves flexibility



Technical Status

- Prototype operational
- External to an ATA/100 drive
 - →performance is currently disk-limited
 - SCSI-based RAID system under development
- 3 applications functional
 - Exact text search
 - Approximate text search (agrep)
 - Biosequence search (Smith-Waterman)



Performance

Speedup relative to 1 GHz processor

Application	Disk-limited speedup	Logic-limited speedup
Exact text search	1.1	14
Approx. text search	12	31
Biosequence search	50	125

Summary

- Fast, inexpensive searches for large and changing databases
- Approximate searches supported
- Up to 100 times faster than standard database searches
- Performance is scalable and uses conventional disk drives
- Data Search Systems, Inc. is actively commercializing the technology



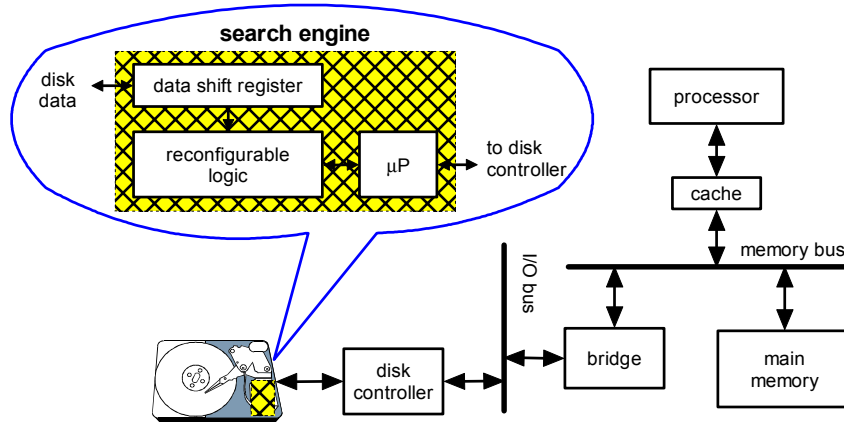


Figure 2. Overall system architecture.

For example, a query might contain the strings “Iraq”, “Iran”, “Israel” and “France”, and want to identify documents where either “Iraq” *and* “Israel”, *or* “Iran” *and* “France” are present. This form of compound inquiry may result in a number of document matches, and the objective is to perform these matches at disk speeds. In general, such queries can be represented as a tree structure where the leaves of the tree correspond to the byte strings being sought (e.g., Iraq, Israel, etc.), and the nodes correspond to the logical operations required of the query. One way of viewing the processing of a compound query is in terms of two components:

- processing of the leaves associated with the bottom of the tree (i.e., taking the leaf words and performing a match with data on the disk), and
- performing the logic operations associated with the combining nodes in the query tree structure.

In this architecture, leaf processing (word matching) is done in the reconfigurable hardware (at disk speeds) and the results are sent to the dedicated control microprocessor that acts to execute the logic associated with the nodes of the query tree. The result of this logic execution is a set of results that are sent to the host processor that initiated system activity by sending the original query.

3. Example Applications

To illustrate the use of the system, a number of example applications are described: unstructured text searches, biological sequence matching, and image searches. All of these applications are currently heavily used, and all tax the capabilities of current systems to deliver the throughput desired.

Unstructured text searching: As described above, a compound query posed in a text search context can be decomposed into the individual word searches (to be executed in the reconfigurable logic) and the composition of the word search results (executed on the

microprocessor). When requiring an exact match at the word level, it is sufficient for the reconfigurable logic to contain a register that stores the string to be matched and a comparator that indicates when the contents of the data shift register equal the string to be matched. We have an implementation of the above design operating with a data throughput of 4 Gb/s (i.e., the system can accept data at that rate sustained).

As an alternative to exact string matches, it is often useful to express the desired search in terms of an acceptable number of mismatches (e.g., insertions, deletions, and substitutions). Algorithms for this type of search (incorporated into the Unix command *agrep*) are described in [2]. Our systolic array implementation of this algorithm executes at a clock rate of 100 MHz, accepting one character each clock, for an aggregate data rate of 800 Mb/s.

Sequence matching: The basic set of operations in genomic or proteomic sequence matching corresponds to a dynamic programming problem when executed on a conventional system [3]. Here, a pattern p is compared against symbols from a target t , and $d_{i,j}$ (an entry in the dynamic programming table) represents the edit distance at position i in the pattern, p_i , and position j in the target, t_j . In normal usage the pattern is short relative to the target. Typical sizes might have p on the order of 1000-2000 characters and t many billions of characters.

The details of our systolic array implementation of the dynamic programming problem are described in [4]. It can sustain a data throughput of 800 Mb/s.

Image searching: Many searching applications operate on data that represent a two-dimensional entity, such as an image. For example, one approach to the object recognition problem is to repeatedly compare the field of interest in an image to templates that store a representation of the objects to be recognized [5]. For imaging applications, the structure of the reconfigurable logic from Figure 2 must take into account the fact that the logical structure of the data is two-dimensional. In addition, the matching operations themselves are often

significantly different on two-dimensional data, and this must also be supported by the reconfigurable logic.

As an example of the type of image search problem that can be accelerated with the proposed system, we have investigated the automatic recognition of objects within synthetic radar (SAR) imagery [5,6]. Here, SAR images are compared with templates in an object database using a conditionally Gaussian data model. An implementation of the reconfigurable logic for this application is under development.

Another potential use of the image search system is object recognition in millimeter wave imagery of people entering a restricted area, such as an airport concourse. In these images, non-metallic sharp objects are readily detected, yet there are serious privacy concerns associated with human scanning, since the modality effectively “sees” through clothing. An automated recognition system that does not present images to operators has the potential to be effective at the security task while mitigating (though not eliminating) the privacy issue.

Other applications: The above section has described the reconfigurable hardware operation for a few specific applications. Other applications will have distinct low-level operations that need to be performed. One strength of this system is the fact that the reconfigurable logic is flexible enough to support not only the example applications already described, but a large variety of additional structures, even those not yet envisioned when the system was designed.

4. Performance

While the overall system is still under development, a sufficient number of component pieces exist to report on their performance. Three reconfigurable hardware search kernels have undergone detailed design (i.e., VHDL-level design), and all are operational. An exact text search engine design operates at 62.5 MHz (eight characters per clock, or 4 Gb/s); the approximate text search engine (agrep) has been tested to 100 MHz (one character per clock, 800 Mb/s); and the biosequence search engine has been tested to 25 MHz (four characters per clock, 800 Mb/s) [4].

Software implementations of these three kernels have been measured on a 933 MHz PC as follows: 280 Mb/s for exact text searches, 26 Mb/s for approximate text searches (allowing up to 8 errors), and 6.4 Mb/s for the biosequence search. Table 1 shows the speedup achievable under two conditions, one is the measured performance as limited by our current ATA (7200 rpm) drive and the other under the conditions where a faster drive is available and the reconfigurable logic kernel is the performance limit.

Table 1. Application speedups.

Application	Disk-limited speedup	Logic-limited speedup
Exact text search	1.1	14
Approx. text search	12	31
Biosequence search	50	125

5. Summary and Conclusions

This paper presents the basic design of a data-mining system that has the potential for truly fast operation, unhindered by the overheads imposed by the I/O bus, main memory bus, cache, operating system, etc. An important requirement for the ultimate success of this system is the decomposition of data search operations into low-level operations that can execute on the reconfigurable hardware and high-level operations that execute in software.

The system achieves performance gains via four mechanisms: reduced data movement overhead, searches operating at hardware speeds, specialization of the hardware logic to the particular query, and parallelism at both the disk and system levels.

We currently have initial prototype implementations of the reconfigurable hardware for several of the applications described above, and are working to further improve their performance. We are also developing performance models that help assess the performance gains that can be achieved using the system.

- [1] John Reynnders, “Computing Biology,” invited talk at 5th High Performance Embedded Computing Workshop, November 2001.
- [2] S. Wu and U. Manber, “Fast text searching allowing errors,” *Communications of the ACM*, **35**(10): 83-91, October 1992.
- [3] Dan Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, 1997.
- [4] Benjamin West, “An FPGA-Based High-Speed Search Engine for Off-the-Shelf Hard Drives,” Master’s thesis, Dept. of Computer Science and Engineering, Washington University, 2003.
- [5] J.A. O’Sullivan, M.D. DeVore, V. Kedia, and M.I. Miller, “SAR ATR performance using a conditionally Gaussian model,” *IEEE Transactions on Aerospace and Electronic Systems*, **37**(1):91-108, January 2001.
- [6] M.D. DeVore, R.D. Chamberlain, G.L. Engel, J.A. O’Sullivan, and M.A. Franklin, “Tradeoffs Between Quality of Results and Resource Consumption in a Recognition System,” in *Proc. of IEEE Int’l Conf. on Application-Specific Systems, Architectures and Processors*, pp. 391-402, July 2002.